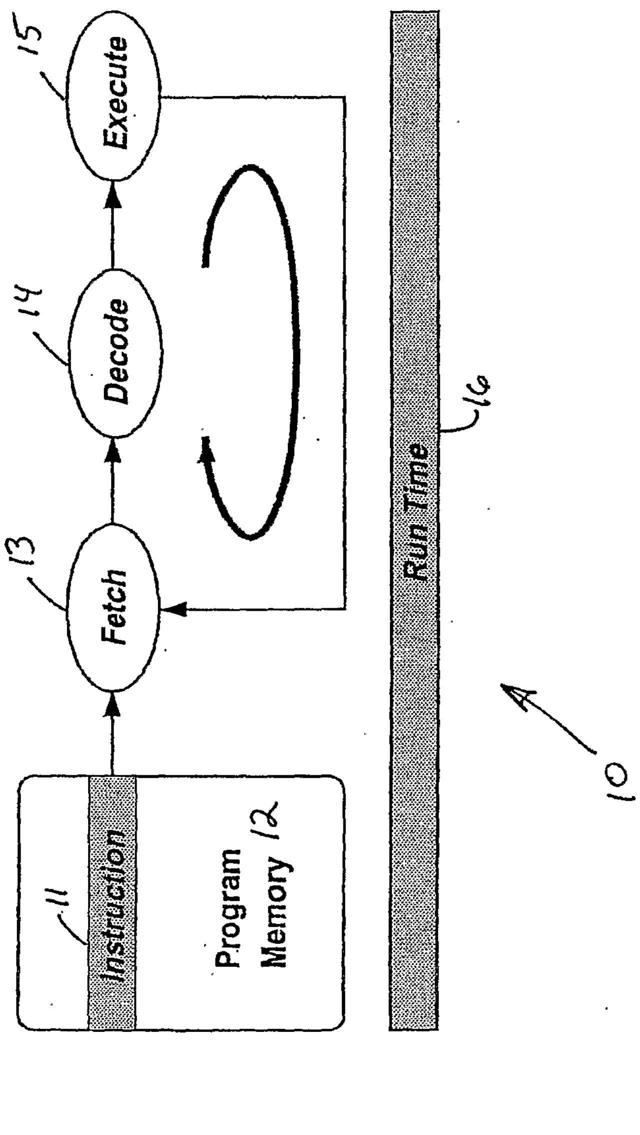


FIG. 1
PRIOR ART

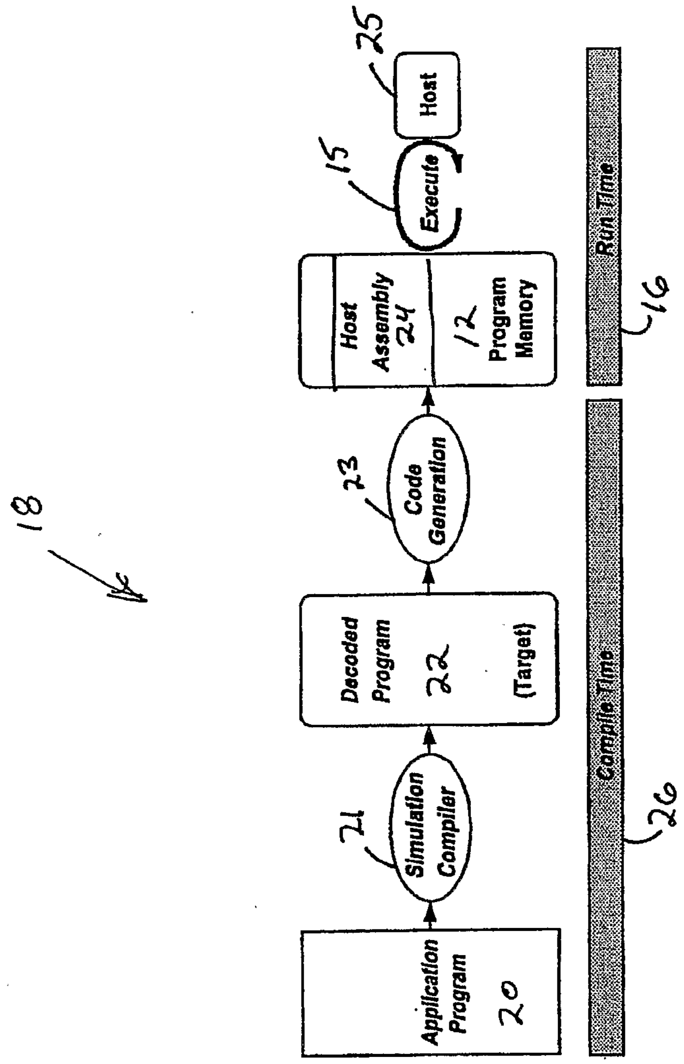


FIG. 2
PRIOR ART

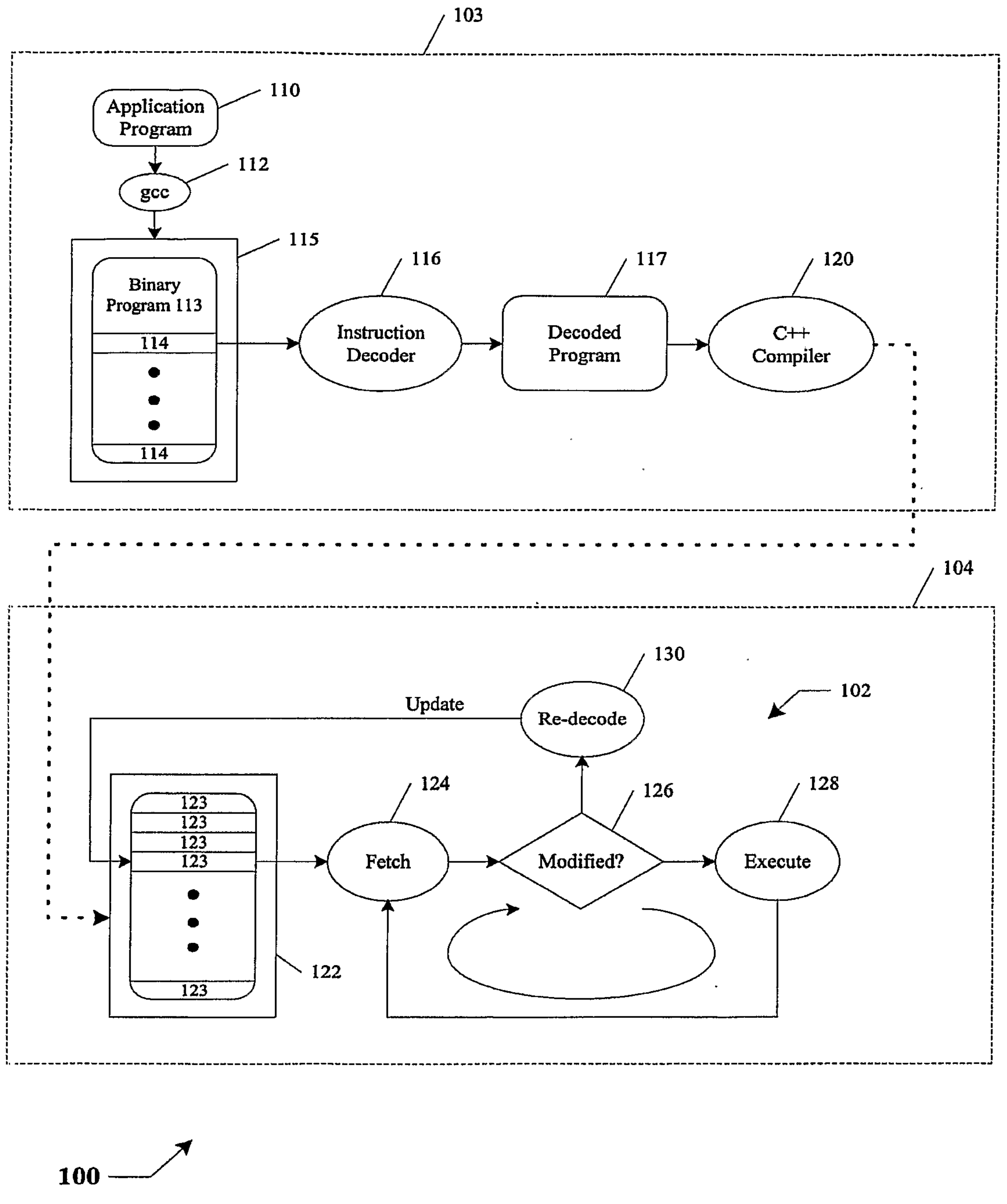
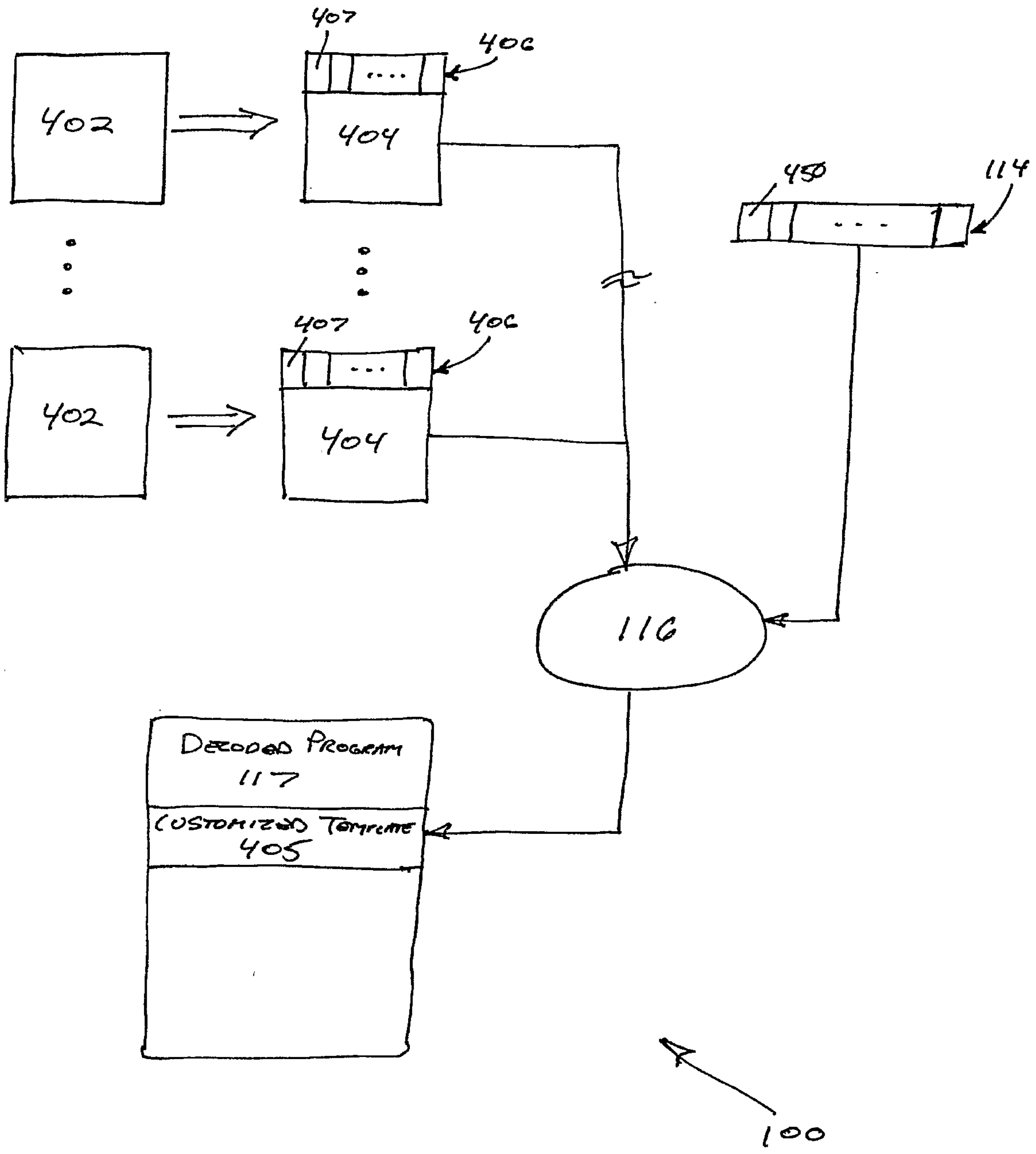

103
Application Program
110
gcc
112
Binary Program 113
114
114
115
Instruction Decoder
116
Decoded Program
117
C++ Compiler
120
104
Update
Re-decode
130
102
123
123
123
123
123
122
Fetch
124
Modified?
126
Execute
128
100

FIG. 3



407
406
402
404
450
114
402
407
406
404
116
DECODED PROGRAM
117
CUSTOMIZED TEMPLATE
405
100

FIG. 4A

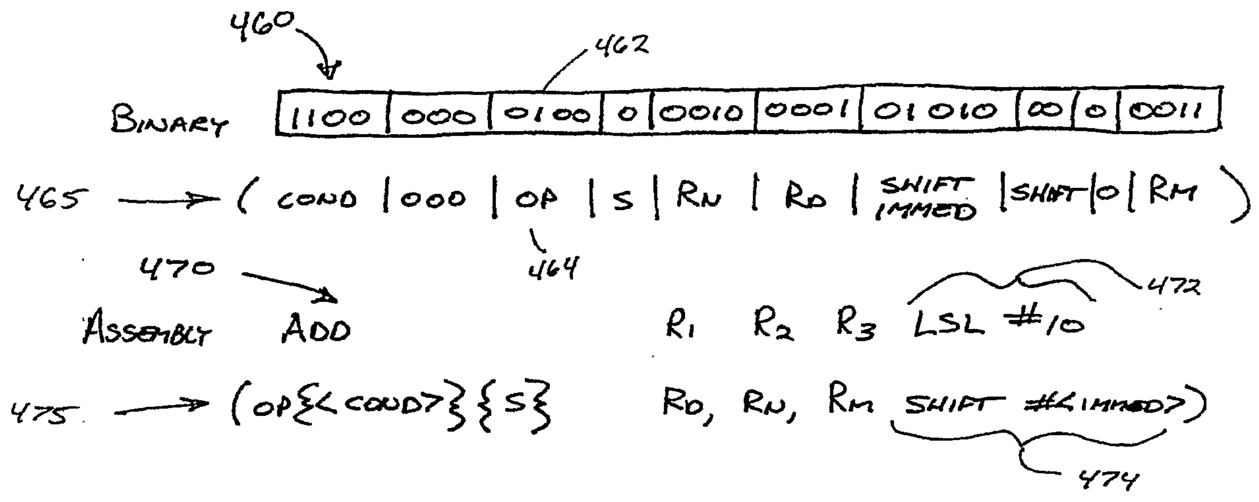

460
462
BINARY
1100 | 000 | 0100 | 0 | 0010 | 0001 | 01010 | 00 | 0 | 0011
465
( COND | 000 | OP | S | RN | RD | SHIFT IMMED | SHIFT | 0 | RM )
464
470
ASSEMBLY ADD R1 R2 R3 LSL #10
472
475
(OP{<COND>}{S} RD, RN, RM SHIFT #<IMMED>)
474

FIG. 4B

```
#pragma once
#ifndef ARMLIB_INSTRUCTION_DATAPROCESSING_HPP_HEADER_INCLUDED
#define ARMLIB_INSTRUCTION_DATAPROCESSING_HPP_HEADER_INCLUDED

#include "SimpressLib/stdIncludes.h"
#include "SimpressLib/instruction/Instruction.hpp"
#include "SimpressLib/val/Reg.hpp"
#include "ARMLib/instruction/lib/Flags.hpp"

namespace ARMLib {
namespace instruction {
template <class Condition, class Operation, class SBit, class RdIsPC, class ShifterOperandType, class RdType, class RnType>
class DataProcessing : public SimpressLib::instruction::Instruction
{
    ShifterOperandType* _shifterOperand;
    RdType* _rd;
    RnType* _rn;

public:
    DataProcessing(ShifterOperandType* shifterOperand, SimpressLib::val::Reg* rd, SimpressLib::val::Reg* rn)
    {
        _shifterOperand = shifterOperand;
        _rd = (RdType*) rd;
        _rn = (RnType*) rn;
    }

    ~DataProcessing(void)
    {
        delete _shifterOperand;
    }

    virtual void execute()
    {
        if (Condition::exec())
        {
            Word lhs = _rn->getWord();
            Word rhs = _shifterOperand->getValue();
            Word result = Operation::exec(lhs, rhs);
            _rd->setWord(result);
            if (SBit::value() == true)
            {
                //update flags!
                int CF = lib::Flags::getC();
                int VF = lib::Flags::getV();

                Operation::updateC(lhs, rhs, result, &CF);
                Operation::updateV(lhs, rhs, result, &VF);

                lib::Flags::setN(NEG(result));
                lib::Flags::setZ((result)? 0 : 1);
                lib::Flags::setC(CF);
                lib::Flags::setV(VF);
            }
        }

    }
};

} //namespace instruction
} //namespace ARMLib
#endif /* ARMLIB_INSTRUCTION_DATAPROCESSING_HPP_HEADER_INCLUDED */
```

404

FIG. 5A

```
#pragma once
#ifndef ARMLIB_INSTRUCTION_LOADSTORE_HPP_HEADER_INCLUDED
#define ARMLIB_INSTRUCTION_LOADSTORE_HPP_HEADER_INCLUDED

#include "SimpressLib/stdIncludes.h"
#include "SimpressLib/instruction/Instruction.hpp"
#include "SimpressLib/val/Reg.hpp"
#include "SimpressLib/component/storage/memory/MainMemory.hpp"

namespace ARMLib {
namespace instruction {
template <class Condition, class LBit, class ElementType, class AddressingMode2_3Type, class RdType>
class LoadStore : public SimpressLib::instruction::Instruction
{
    AddressingMode2_3Type* _addressingMode2_3;
    RdType* _rd;
    SimpressLib::component::storage::memory::MainMemory* _mainMemory;
public:
    LoadStore(AddressingMode2_3Type* addressingMode2_3, SimpressLib::val::Reg* rd)
    {
        _addressingMode2_3 = addressingMode2_3;
        _rd = (RdType*) rd;
        _mainMemory = ARCH_CALL(getMainMemory());
    }

    ~LoadStore(void)
    {
        delete _addressingMode2_3;
    }

    virtual void execute()
    {
        if (Condition::exec())
        {
            Word addr = _addressingMode2_3->getAddress();
            if (LBit::value() == true)
            {//Load
                if ((addr % sizeof(ElementType::type)) !=0)
                    FATAL_ERROR("Invalid address for loading!");

                int elemValue = (int) _mainMemory->load<ElementType::type>(addr);
                _rd->setWord(elemValue);
            }
            else
            {//Store
                ElementType::type elemValue = (ElementType::type) _rd->getWord();
                _mainMemory->store<ElementType::type>(addr, elemValue);
            }
        }

    }
};

} //namespace instruction
} //namespace ARMLib
#endif /* ARMLIB_INSTRUCTION_LOADSTORE_HPP_HEADER_INCLUDED */
```

404

FIG. 5B

```
#pragma once
#ifndef ARMLIB_INSTRUCTION_MULTILPY_HPP_HEADER_INCLUDED
#define ARMLIB_INSTRUCTION_MULTILPY_HPP_HEADER_INCLUDED

#include "SimpressLib/stdIncludes.h"
#include "SimpressLib/instruction/Instruction.hpp"
#include "SimpressLib/val/Reg.hpp"
#include "ARMLib/instruction/lib/Flags.hpp"

namespace ARMLib {
namespace instruction {
template <class Condition, class SBit, class AccumulateBit>
class Multiply : public SimpressLib::instruction::Instruction
{
private:
        SimpressLib::val::Reg* _rd;
        SimpressLib::val::Reg* _rn;
        SimpressLib::val::Reg* _rs;
        SimpressLib::val::Reg* _rm;
public:
        Multiply(SimpressLib::val::Reg* rd, SimpressLib::val::Reg* rn, SimpressLib::val::Reg* rs, SimpressLib::val::Reg* rm)
        {
                _rd = rd;
                _rn = rn;
                _rs = rs;
                _rm = rm;
        }

        ~Multiply(void)
        {
        }

        virtual void execute()
        {
                if (Condition::exec())
                {
                        Word result = _rm->getWord() * _rs->getWord();
                        if (AccumulateBit::value())
                                result += _rn->getWord();

                        _rd->setWord(result);
                        if (SBit::value() == true)
                        {
                                lib::Flags::setN(NEG(result));
                                lib::Flags::setZ((result)? 0 : 1);
                        }
                }

        }
};

} //namespace instruction
} //namespace ARMLib
#endif /* ARMLIB_INSTRUCTION_MULTILPY_HPP_HEADER_INCLUDED */
```

404

FIG. 5C

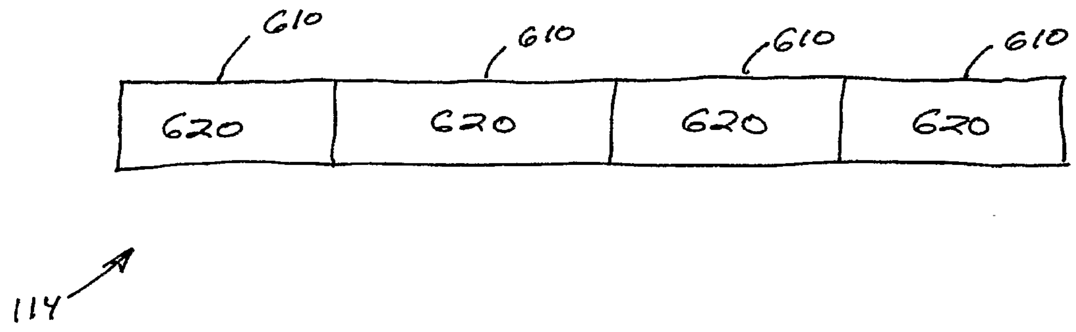


FIG. 6A

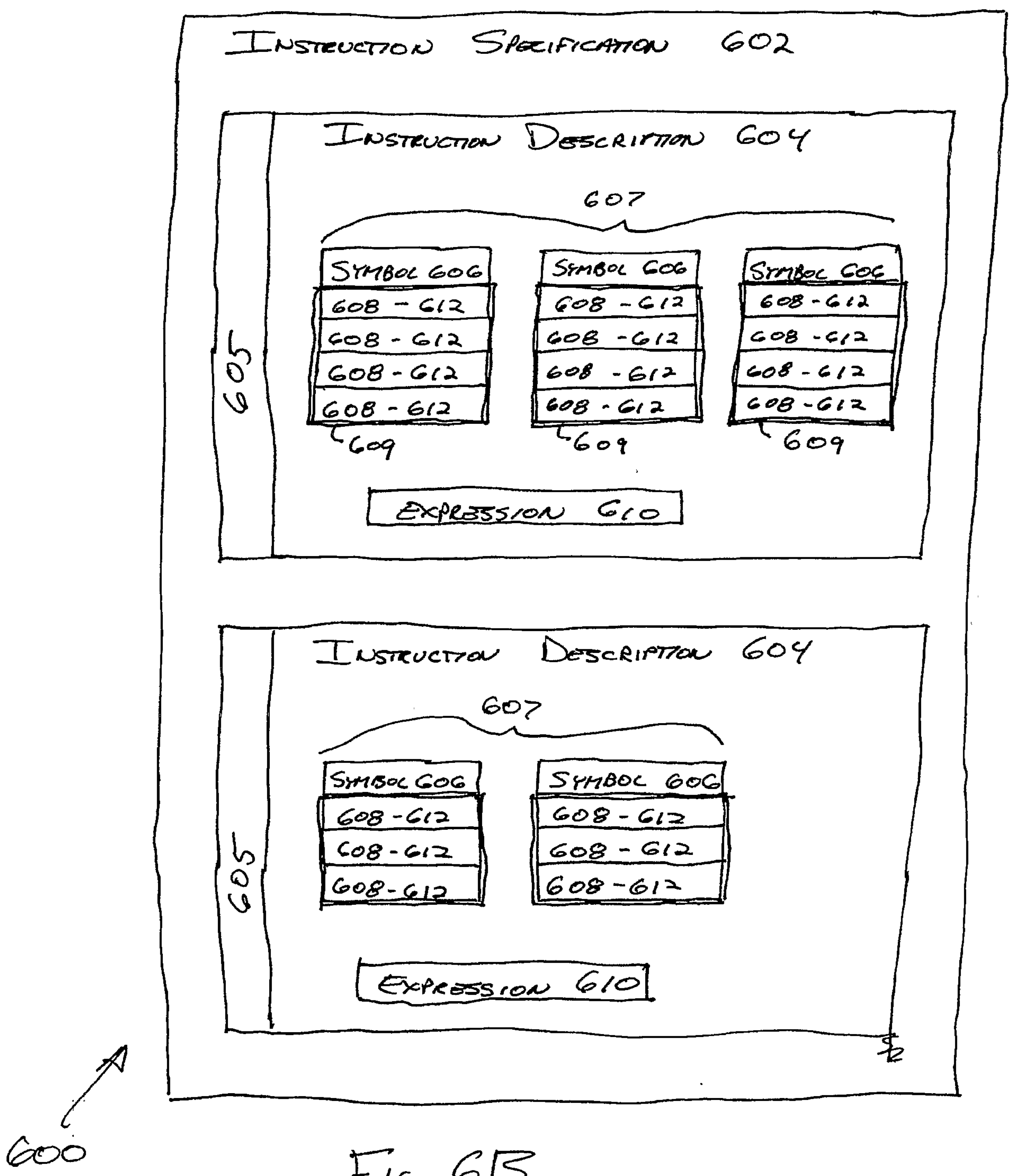

INSTRUCTION SPECIFICATION 602
INSTRUCTION DESCRIPTION 604
607
SYMBOL 606
608 - 612
609
605
EXPRESSION 610
600

FIG. 6B

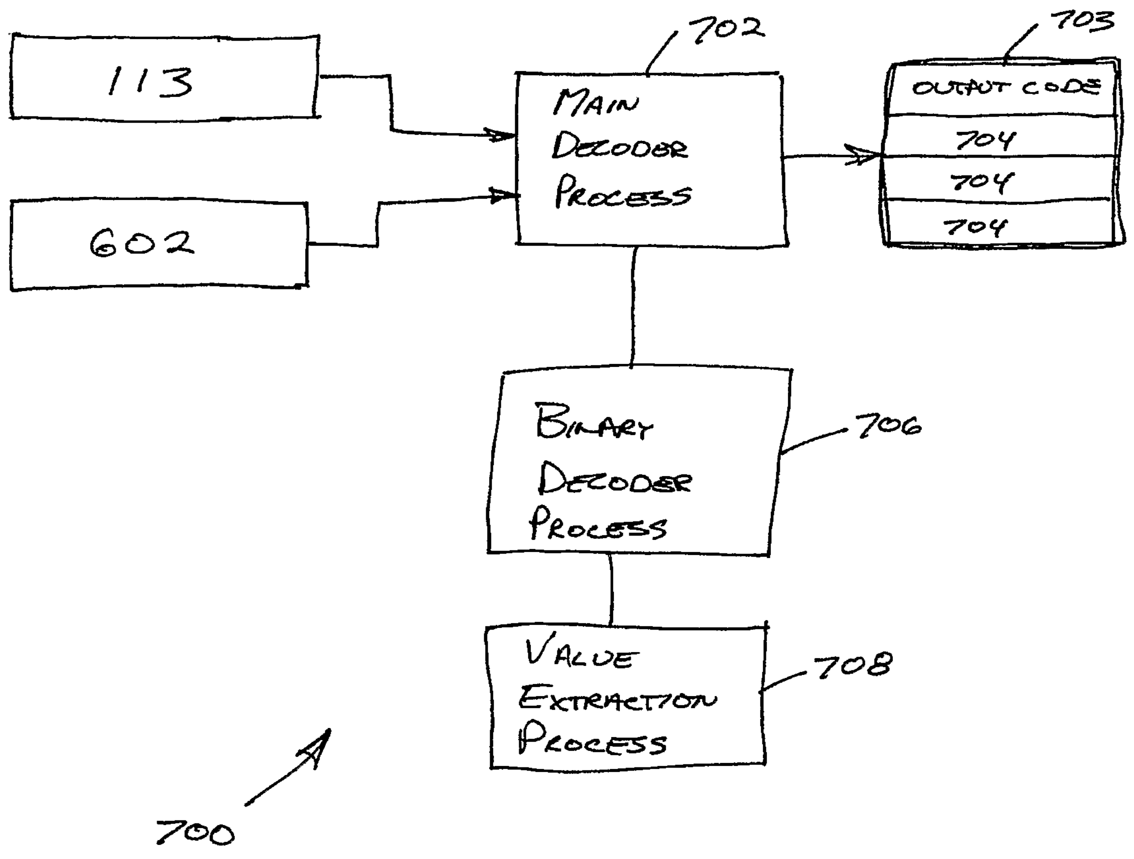

113
602
MAIN DECODER PROCESS
702
703
OUTPUT CODE
704
704
704
BINARY DECODER PROCESS
706
VALUE EXTRACTION PROCESS
708
700

FIG. 7

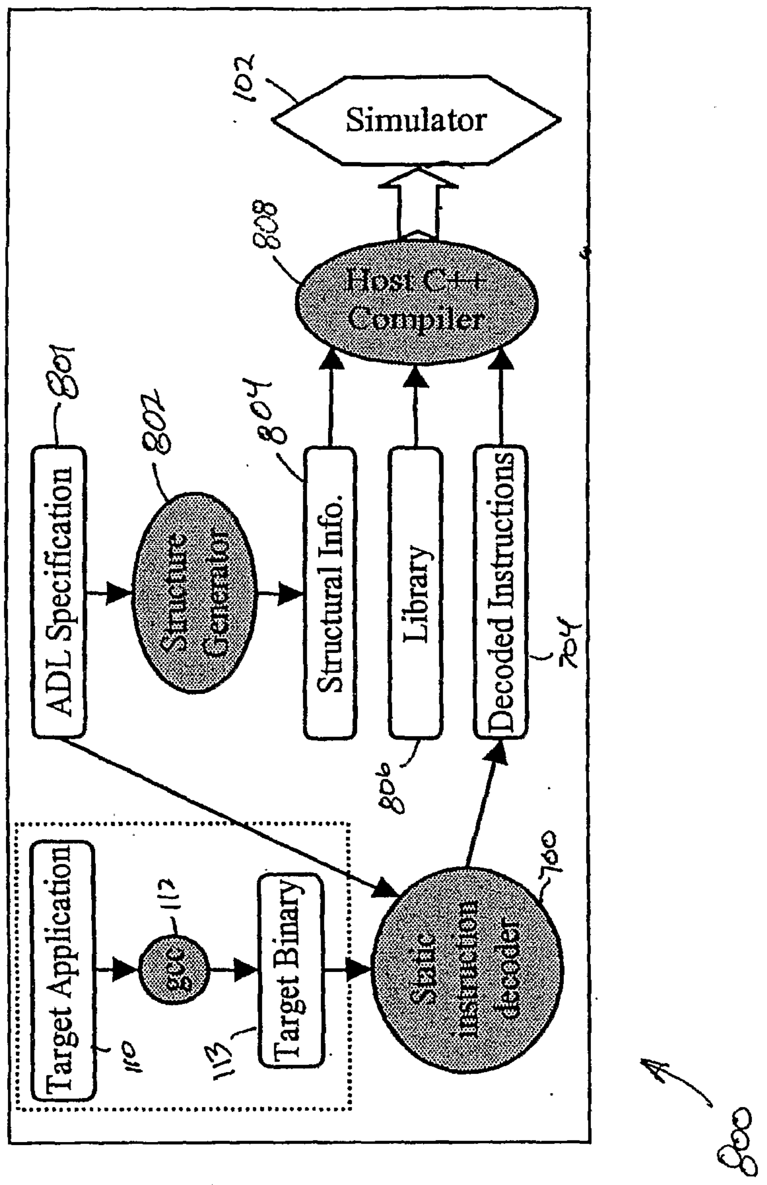

Target Application
110
gcc
112
Target Binary
113
ADL Specification
801
Structure Generator
802
Structural Info.
804
Library
806
Decoded Instructions
704
Static instruction decoder
700
Host C++ Compiler
808
Simulator
102
800

FIG. 8